An Early Performance Evaluation of Many Integrated Core Architecture Based SGI Rackable Computing System

Subhash Saini,[1] Haoqiang Jin,[1] Dennis Jespersen,[1] Huiyu Feng,[2] Jahed Djomehri,[3] William Arasin,[3] Robert Hood,[3] Piyush Mehrotra,[1] Rupak Biswas[1]

[1]NASA Ames Research Center
Moffett Field, CA 94035-1000, USA
{subhash.saini, haoqiang.jin, dennis.jespersen, piyush.mehrotra, rupak.biswas}@nasa.gov

[2]SGI
Fremont, CA 94538, USA
hfeng@sgi.com

[3]Computer Sciences Corporation
Moffett Field, CA 94035-1000, USA
{jahed.djomehri, william.f.arasin, robert.hood}@nasa.gov

ABSTRACT

Intel recently introduced the Xeon Phi coprocessor based on the Many Integrated Core architecture featuring 60 cores with a peak performance of 1.0 Tflop/s. NASA has deployed a 128-node SGI Rackable system where each node has two Intel Xeon E2670 8-core Sandy Bridge processors along with two Xeon Phi 5110P coprocessors. We have conducted an early performance evaluation of the Xeon Phi. We used microbenchmarks to measure the latency and bandwidth of memory and interconnect, I/O rates, and the performance of OpenMP directives and MPI functions. We also used OpenMP and MPI versions of the NAS Parallel Benchmarks along with two production CFD applications to test four programming modes: offload, processor native, coprocessor native and symmetric (processor plus coprocessor). In this paper we present preliminary results based on our performance evaluation of various aspects of a Phi-based system.

Categories and Subject Descriptors

H.5.2 **[Information Interfaces and Presentation]:** User Interfaces – benchmarking, evaluation/methodology

General Terms

Measurement, Performance, and Experimentation

Keywords

Intel Xeon Phi, Intel MIC architecture, Intel Sandy Bridge processor, performance evaluation, benchmarking, CFD applications

1. INTRODUCTION

The stagnation of processor frequency due to the constraints of power and current leakage has led hardware vendors to increase parallelism in their processor designs in order to enhance the performance of highly parallel scientific and engineering applications. This has led to an era of heterogeneous computing where highly parallel accelerators are paired with modestly parallel x86-compatible processors. The two current approaches use either the NVIDIA's General-Purpose Graphical Processing Unit (GPGPU) or the Intel Xeon Phi.

(c) 2013 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

SC '13, November 17 - 21 2013, USA
Copyright 2013 ACM 978-1-4503-2378-9/13/11…$15.00.
http://dx.doi.org/10.1145/2503210.2503272

The GPGPU approach relies on streaming multiprocessors and uses a low-level programming model such as CUDA or a high-level programming model like OpenACC to attain high performance [1-3]. Intel's approach has a Phi serving as a coprocessor to a traditional Intel processor host. The Phi has x86-compatible cores with wide vector processing units and uses standard parallel programming models such as MPI, OpenMP, hybrid (MPI + OpenMP), UPC, etc. [4-5].

Understanding performance of these heterogeneous computing systems has become very important as they have begun to appear in extreme-scale scientific and engineering computing platforms, such as Blue Waters at NCSA, Stampede at TACC, and Titan at Oak Ridge National Laboratory (ORNL) [6-8]. Large-scale GPGPU-based supercomputers have been around for the last ten years and a significant amount of research work—including applications, algorithms and performance evaluation—has been done, resulting in a vast amount of research literature on the subject. However, the relative newness of the Many Integrated Core (MIC)—it has its genesis with the Larabee project in 2006—means that there is a dearth of scientific literature on using it to achieve high performance in scientific and engineering applications [8-15].

In this paper we study the performance of "Maia," a 128-node InfiniBand-connected cluster, where each node has two Intel Xeon E5-2670 (Sandy Bridge) processors and two Xeon Phi 5110P coprocessors. In the rest of the paper we refer to the two Sandy Bridge processors collectively as the "host" and the Xeon Phi coprocessors as the "Phi", using "Phi0" and "Phi1" when we need to distinguish between the two coprocessors. Several papers have reported on the experience of porting applications to the non-commercial version of the MIC (32-core Knights Ferry chips) in MIC workshops at TACC and ORNL [11-12]. However, none of the reports at these two workshops gave any performance numbers or carried out a detailed performance evaluation of the Intel Xeon Phi product. To the best of our knowledge the following is our original contribution:

a. We measured the latency and memory bandwidth of L1, L2, caches, and main memory of Phi. In addition, we also measured the Peripheral Component Interconnect Express (PCIe) latency and bandwidth achieved by the MPI and offload modes between host and Phi on the same node.

b. We measured and compared the performance of intra-node MPI functions (point-to-point, one-to-many, many-to one, and all-to-all) for both host and Phi.

c. We measured and compared the overhead of OpenMP constructs for synchronization, loop scheduling, and data movement on host and Phi.

d. We measured and compared the I/O performance for both the host and Phi.

e. We measured and compared the performance of MPI and OpenMP based NAS Parallel Benchmarks (NPBs) 3.3 on both the host and Phi.

f. We measured and compared the performance of two production-level applications using different modes: native host, native Phi and symmetric (host+Phi0+Phi1) modes.

The remainder of the paper is organized as follows. Section 2 provides details of the Maia computing system. In Section 3 we briefly describe the benchmarks and applications used in the current study. Section 4 describes the four programming models available on Phi-based heterogeneous computing systems. Section 5 gives the description of pre-update and post-update software environments. In Section 6 we discuss the performance results obtained in our evaluation of the Sandy Bridge processors based host and the Phi coprocessors. In Section 7 we present our conclusions.

2. COMPUTING PLATFORM "MAIA"

The NASA Advanced Supercomputing (NAS) Division at NASA Ames recently installed a 128-node heterogeneous SGI Rackable computing system called "Maia." Each node has two Intel Xeon E5-2670 (Sandy Bridge) processors and two Intel Xeon Phi coprocessors. The Sandy Bridge is an 8-core processor using a 32-nm process technology. Each Phi is a 60-core Symmetric Multi Processor (SMP) on a chip and uses 22-nm process technology. Overall, the system has a theoretical peak performance of 301.4 Tflop/s. Of that peak performance, 42.6 Tflop/s comes from the 2,048 Sandy Bridge cores and 258.8 Tflop/s comes from the 15,360 Phi cores. The system has 4 TB of memory available to the Sandy Bridge processors and 2 TB for the Phi coprocessors for a total memory of 6 TB.

Maia's heterogeneous architecture is depicted in Figure 1. Each of the 128 nodes has three distinct memory systems. The main host memory is 32 GB and is shared in a cache coherent fashion by the 16 cores of the two Sandy Bridge processors. The cores in each of the Phi coprocessors share an 8-GB cache-coherent memory system. Each Phi is connected to other devices on the node via a separate 16-lane PCI Express (PCIe) bus [4-5]. Connectivity to other nodes is provided by a fourteen data rate (FDR) InfiniBand Host Channel Adapter (HCA) plugged into the first PCIe bus [31].

The Phi coprocessors run a BusyBox-based micro Linux operating system. A virtualized TCP/IP stack is implemented over PCIe, permitting access to the coprocessors as network nodes. This facilitates connecting to each coprocessor through a secure shell (ssh) and running jobs. The two Phis within a node can communicate with each other via the PCIe peer-to-peer interconnect with intervention from the host processor.

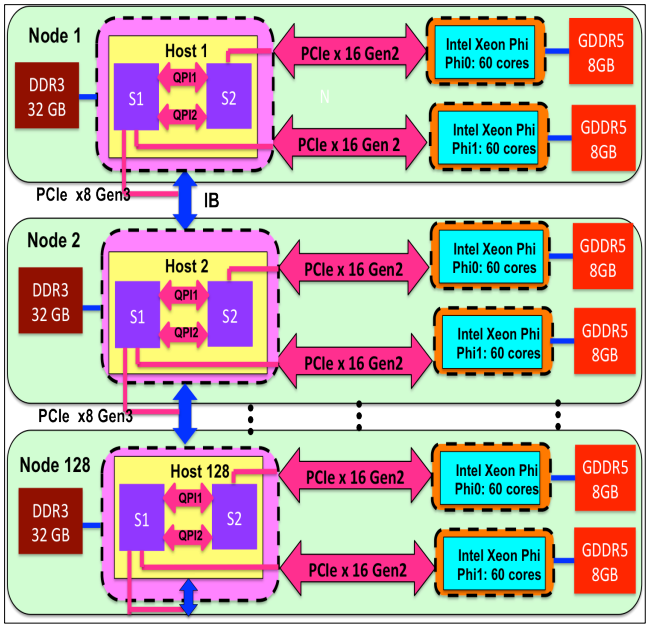

Figure 1. Maia's heterogeneous architecture.

As shown in Figure 2, the Sandy Bridge based node has two Xeon E5-2670 processors. Each processor has eight cores, clocked at 2.6 GHz, with a peak performance of 166.4 Gflop/s [15]. Each core has 64 KB of L1 cache (32 KB data and 32 KB instruction) and 256 KB of L2 cache. All eight cores share 20 MB of last level cache, also called L3 cache. The on-chip memory controller supports four DDR3 channels running at 1600 MHz, with a peak-memory bandwidth per processor of 51.2 GB/s. Each processor has two QPI links to connect with the second processor of a node to form a non-uniform-memory access (NUMA) architecture. Each QPI link runs at 8 GT/s ("T" for transactions), at which rate 2 bytes can be transferred in each direction, for an aggregate rate of 32 GB/s.

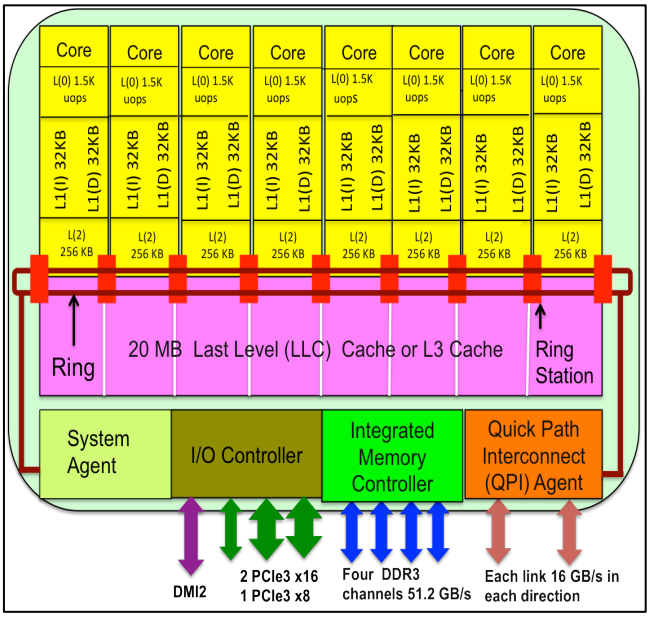

Figure 2. Schematic diagram of the Intel Xeon E5-2670 "Sandy Bridge" processor.

2.1 Coprocessor: Intel Xeon Phi

Figure 3 shows the schematic diagram of a Phi coprocessor. Each Phi coprocessor has 60 modified P54C-design in-order execution cores running at 1.05 GHz and connected by a bi-directional ring interconnect. The core architecture is based on the x86 Instruction Set Architecture extended with 64-bit addressing and new 512-bit wide SIMD vector instructions and registers. Each core can execute two instructions per clock cycle. Also, a core has 32-KB, 8-way set-associative L1(I) and L1(D) caches and a 512-KB unified L2 cache. The L2 caches are connected to the Core Ring Interface, which is used to make memory requests. The L2 caches are kept coherent by a globally distributed tag directory (TD) hanging off the ring. Each Phi core has a dedicated 512-bit wide vector floating-point unit unlike a Sandy Bridge core, which has a 256-bit wide floating-point unit. However, Phis do not support MMX, SSE or AVX instructions.

Each Phi coprocessor has 8 memory controllers, which support Graphics Double Data Rate, version 5 (GDDR5) channels. Each controller can operate two 32-bit channels for a total of 16 memory channels that are capable of delivering 5 GT/s per channel. These memory controllers are interleaved around the ring symmetrically. Tag directories have an all-to-all mapping to the eight memory controllers.

Each Phi core supports four hardware threads for a total of 240 hardware threads in one Phi coprocessor. Multithreading on MIC is entirely different from the HyperThreading (HT) on the Sandy Bridge architecture. In Sandy Bridge, the aim of HT is to exploit the processor resources more efficiently, whereas in MIC it is to hide latencies inherent in an in-order microarchitecture [16-17]. HT can be turned on or off on a Sandy Bridge processor but multithreading cannot be turned off on a Phi. In addition, compute intensive applications don't benefit (and rather may be hurt) using HT on Sandy Bridge, whereas applications benefit by using multithreading on the Phi. There are four thread contexts per physical core. Registers are replicated, but L1 and L2 caches are shared among the threads in a core. When one thread stalls, the processor makes a context switch to another one. However, it cannot issue back-to-back instructions in the same thread.

The MIC architecture has a provision for high performance reciprocal, square root, power, and exponent operations, as well as scatter/gather and streaming store capabilities for high memory bandwidth.

Table 1 gives the detailed hardware and software characteristics of the Maia system.

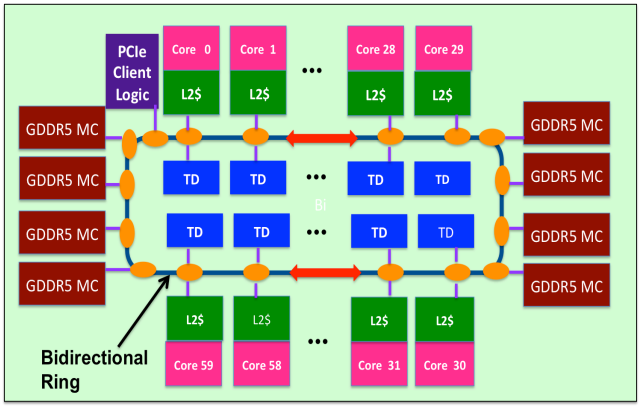

Figure 3. Schematic diagram of a Phi coprocessor.

TABLE 1. CHARACTERISTICS OF MAIA, SGI RACKABLE C1104G-RP5 SYSTEM.

	Host Processor	Coprocessor
Processor		
Processor architecture	Sandy Bridge	Many Integrated Core
Processor type	Intel Xeon E5-2670	Intel Xeon Phi 5110P
Number cores/processor	8	60
Base frequency (GHz)	2.60	1.05
Turbo frequency (GHz)	3.20	NA
Floating points / clock	8	16
Perf. /core (Gflop/s)	20.8	16.8
Proc. perf. (Gflop/s)	166.4	1008
New instruction	SSE4.1, 4.2 and AVX	MIC VEC Instruction
SIMD vector width	256	512
Number of threads / core	2	4
Multithreading on/off	On or Off	Always ON
Type of multithreading	HyperThread	Hardware Threads
I/O controller	On chip	NA
Cache		
L1 cache size / core	32 KB (I)+32 KB (D)	32 KB (I)+32 KB (D)
L2 cache size / core	256 KB	512 KB
L2 cache network	Bi-directional ring	Bi-directional ring
L3 cache size	20 MB (shared)	NA
L3 cache network	Bi-directional ring	NA
Node		
No. of processors/node	2	2
QPI frequency (GT/s)	8.0	NA
Number of QPIs	2	NA
Memory type	4 channels DDR3-1600	GDDR5-3400
Memory / node (GB)	32	16 GB-8 GB / Phi card
Sock-sock interconnect	2 QPIs, 8.0 GT/s	NA
Host-Phi interconnects	PCIe	PCIe
PCI Express	40 Integrated PCIe 3.0	16 Integrated PCIe 2.0
PCIe Speed	8 GT/s (Gen3)	5 GT/s (Gen2)
System		
Number of nodes	128	
Total cores	2048	15360
Peak perf. (Tflop/s)	42.6	258
% Flops	14	86
Interconnect type	4x FDR InfiniBand	
Peak network perf.	56 GB/s	
Network topology	Hypercube	
Type of file system	Lustre	
Software		
Compiler	Intel 13.1	
MPI library	Intel MPI 4.1	
Math library	Intel MKL 10.1	
Operating system	SLES11SP2	MPSS Gold

3. BENCHMARKS AND APPLICATIONS

In this section we present a brief description of the benchmarks and applications used in this paper

3.1 STREAM Benchmark

Performance of many applications depends on the memory bandwidth so it is important to measure it. STREAM is a simple, synthetic benchmark program that measures sustainable memory bandwidth for simple vector kernels such copy, add, BLAS1, etc. We used STREAM version 5.1 [18].

3.2 Memory Subsystem Latency and Bandwidth

Deep understanding of the performance of the hierarchical memory system of the Sandy Bridge host and of the Phi coprocessors is a crucial to obtain good application performance.

We measured the latency and bandwidth for all caches and main memory for both the host and the Phi [19-20].

3.3 MPI Functions Benchmarks

The performance of real-world applications that use MPI as the programming model depends significantly on the MPI library and the performance of various point-to-point and collective message exchange operations. The MPI standard defines several collective operations, which can be broadly classified into three major categories based on the message exchange pattern: OnetoAll, AlltoOne, and AlltoAll. We have measured and evaluated the performance of MPI_Bcast*, MPI_Send, MPI_Recv, MPI_AllGather, MPI_AlltoAll* and *MPI_Allreduce* functions on both host and coprocessor [21].

3.4 OpenMP Microbenchmarks

The OpenMP microbenchmarks are a set of tests to measure the overheads of various OpenMP directives and constructs in dealing with loop scheduling, synchronization, and data privatization. These benchmarks measure the overhead of OpenMP directives by subtracting the time for executing the code sequentially from the time taken by the same code executed in parallel enclosed in a given directive [22-24].

3.5 Sequential I/O Benchmark

I/O is critical for overall performance of the application. Real world applications have large amounts of data (100's of GB to 100's of TB) to read and write. I/O is also important as most of the applications perform checkpointing, which requires fast writes. Sequential Read Write is a single process I/O benchmark that writes and reads a file using various block sizes [25].

3.6 NAS Parallel Benchmarks (NPB)

The NPB suite contains eight benchmarks comprising five kernels (CG, FT, EP, MG, and IS) and three compact applications (BT, LU, and SP) [21]. We used the MPI and OpenMP versions (NPB 3.3), Class C problem in our study. BT, LU, and SP are typical of full, production-quality science and engineering applications [26].

3.7 Science and Engineering Applications

For this study, we used two production quality full applications representative of NASA's and aerospace companies workload.

3.7.1 OVERFLOW-2

OVERFLOW-2 is a general-purpose Navier-Stokes solver for CFD problems [27-28]. The code uses finite differences in space with implicit time stepping. It uses overset-structured grids to accommodate arbitrarily complex moving geometries. The dataset used is a wing-body-nacelle-pylon geometry (DLRF6-Large) with 23 zones and 35.9 million grid points. The input dataset is 1.6 GB in size, and the solution file is 2 GB. We also used a smaller data set (DLRF6-Medium) with 10.8 million grid points, as the DLRF6-Large case is too large to run on a single Phi coprocessor.

3.7.2 Cart3D

Cart3D is a high fidelity, inviscid CFD application that solves the Euler equations of fluid dynamics [28-30]. It includes a solver called Flowcart, which uses a second-order, cell-centered, finite volume upwind spatial discretization scheme, in conjunction with a multi-grid accelerated Runge-Kutta method for steady-state cases. In this study, we used the OneraM6 wing with 6 million grid points.

4. PROGRAMMING MODES

On Maia the following four programming modes are available to run the applications. In this paper we have evaluated all four.

4.1 Offload

In this mode, an application is launched on the host, and then parallel compute-intensive subroutines/functions are "offloaded" to the Phi. This is achieved by using "offload" directives, which take care of code execution and data transfer seamlessly. The program specifies what data or subroutine gets offloaded to Phi. The offload directives are followed by one or more OpenMP parallel region to distribute work over Phi threads. Efficiency of this mode depends on how much work can be done on the Phi to offset the cost of the data transfer.

4.2 Native Host

In this mode, the entire application is run exclusively on the host Sandy Bridge processors; Phi coprocessors are not used.

4.3 Native Phi

In this mode, the entire application runs only on the Phi coprocessors. Applications with significant serial regions will suffer dramatically because of the relatively slow clock rate of a Phi core. OpenMP parallel regions will run on Phi cores. MPI codes can be run in a similar way. In many cases, a code that runs fine on the host can be compiled and built without any changes. However, to get even a reasonable performance on the Phi, an application has to be highly parallel and highly vectorized with unit stride. If an application has non-unit memory strides involving gather/scatter its performance degrades dramatically.

4.4 Symmetric

In this mode, an application is run using both the host processors and the Phi coprocessors; it needs to be compiled for host and Phi separately. The challenge is to optimally load balance the work between the host and coprocessors. Hybrid programming (MPI + OpenMP) is more appropriate for this mode. Pure MPI applications can be run but the performance of communication intensive applications would be degraded due to low network communication bandwidth via PCIe to the host or to another Phi.

5. SOFTWARE UPDATE

As early adopters of the new Phi coprocessor, we were faced with an evolving software environment during the course of this evaluation. Initially, we utilized Intel's Manycore Platform Software Stack "MPSS Gold" version and Intel MPI library version 4.1.0.030, henceforth called "pre-update" software. By the end of the study, the software environment had been upgraded to the "MPSS Gold update 3" and MPI library version 4.1.1.036, henceforth called "post-update" software.

The new MPI library in the post-update software switches between different Direct Access Programming Library (DAPL) providers based on message size. For smaller messages, Intel recommends using a Coprocessor Communications Links (CCL) Direct DAPL provider, such as ofa-v2-mlx4_0-1, because it has the lowest latency data path and is available across all network segments. For larger messages, Intel recommends using the Symmetric Communication Interface (SCIF) DAPL provider, ofa-v2-scif0, due to its higher bandwidth data path over the PCIe bus. The pre-update software uses the CCL Direct DAPL provider for all message sizes. In order to use the automatic switching capability in the post-update software, we set two environment variables to specify which DAPL providers are used for various message sizes. Specifically, we used:

```
I_MPI_DAPL_DIRECT_COPY_THRESHOLD=8192,262144
I_MPI_DAPL_PROVIDER_LIST=ofa-v2-mlx4_0-1,ofa-v2-scif0
```

This results in three states:

- Messages shorter than or equal to 8 KB use the "eager protocol" through the CCL direct DAPL provider.
- Messages larger than 8 KB, but shorter than or equal to 256 KB, use the "rendezvous direct-copy protocol" through the CCL direct DAPL provider.
- Messages larger than 256 KB use the rendezvous direct-copy protocol through the DAPL over the SCIF provider.

It should be noted that post-upgrade software does not affect the MPI performance of the native Phi mode or native host mode, which account for most of the benchmarking results in this paper. Only MPI latency over PCIe, MPI bandwidth over PCIe, and the OVERFLOW performance in a symmetric mode would be affected by the settings for DAPL over PCIe. When those results are presented, we describe the performance impact of the update.

6. RESULTS

In this section we present our results for low-level benchmarks, NPBs, and two full applications.

6.1 STREAM Triad Memory Benchmark

Figure 4 shows the total STREAM triad memory bandwidth for both the host and Phi0. We found a maximum aggregate memory bandwidth of 180 GB/s for the Phi using 59 threads (1 thread per core) and 118 threads (2 threads per core). Beyond 118 threads it drops to 140 GB/s. The plausible reason for the drop is that there are more independent memory access streams than there are simultaneously active pages. GDDR5 supports 16 independent banks per device and with eight devices it amounts to 128 open banks, which cause the bandwidth to drop beyond 128 threads.

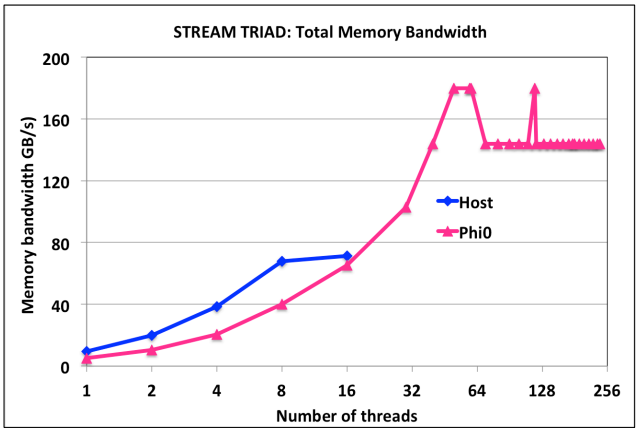

Figure 4. STREAM bandwidth for host and Phi.

6.2 Memory Load Latency and Bandwidth

In this section, we present memory load latency and bandwidth of all caches and main memory on both host and Phi. The total cache per core on a Phi is 544KB (32KB L1 + 512KB L2), which is lower than the 2.788 MB (32KB L1 + 256KB L2 + 2.5MB L3) on the host by a factor of 5.1.

6.2.1 Memory Latency

Figure 5 shows the measured memory latency for both host and Phi. For the host there are four distinct regions corresponding to L1 (32KB), L2 (256KB), L3 (20 MB) cache and main memory (> 20 MB) with latencies of 1.5 ns, 4.6 ns, 15 ns, and 81 ns respectively. Similarly, for the Phi there are three such regions: L1 (32KB), L2 (512KB) and main memory (> 512 KB) with latencies of 2.9 ns, 22.9 ns, and 295 ns respectively.

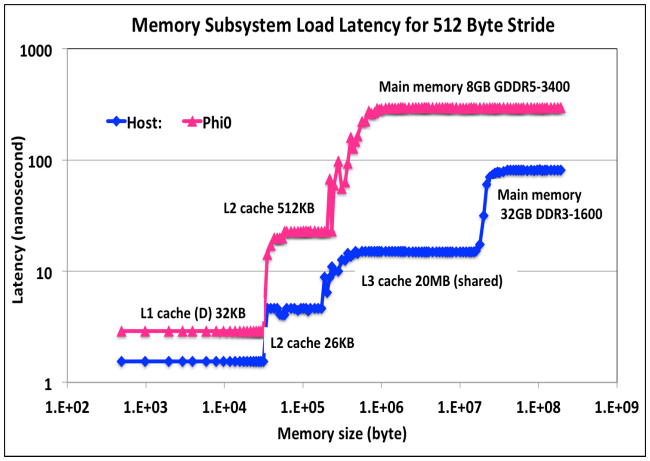

Figure 5. Memory load latency for host and Phi.

6.2.2 Memory Bandwidth

Figure 6 shows the read and write memory load bandwidth per core for the host and Phi. Here also there are four regions (L1, L2, L3, and main memory) on the host and three regions (L1, L2, and main memory) on the Phi. For the four regions on the host, write bandwidths are 10.4, 9.5, 8.6, and 7.2 GB/s; read bandwidths are 12.6, 12.3, 11.6, and 7.5 GB/s, respectively. For the three regions on the Phi, write bandwidths are 1538, 962, and 263 MB/s; read bandwidths are 1680, 971, and 504 MB/s, respectively.

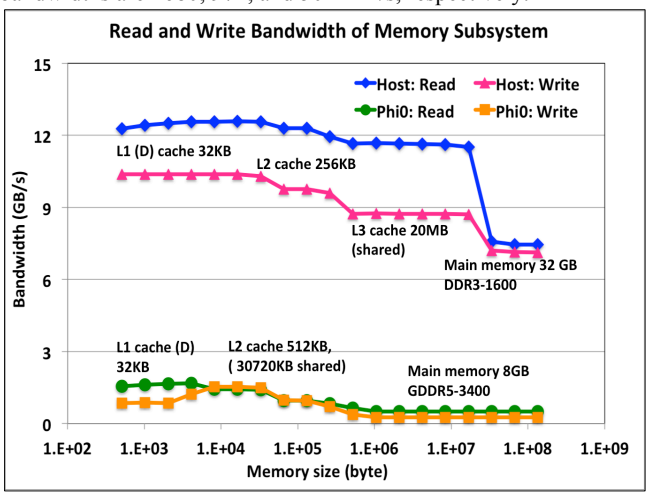

Figure 6. Read and write memory load bandwidth per core for host and Phi.

6.3 Network Latency and Bandwidth

In this subsection we present the interconnect MPI latency and MPI bandwidth between different components on a single node for both pre-update and post-update software. Figures 7 and 8 plot the interconnect latency and bandwidth respectively for connections from: host to Phi0, host to Phi1, and Phi0 to Phi1.

6.3.1 MPI Latency

Latency with pre-update software was 3.3 µs, 4.6 µs, and 6.3 µs for host to Phi0, host to Phi1, and host to Phi0 to Phi1 respectively. The corresponding numbers with post-update software are 3.3 µs, 4.1 µs, and 6.6 µs. It should be noted that latency with both pre-update and post-update software is almost same. However, latencies in the cases involving Phi1 are much higher than the one where only Phi0 is involved.

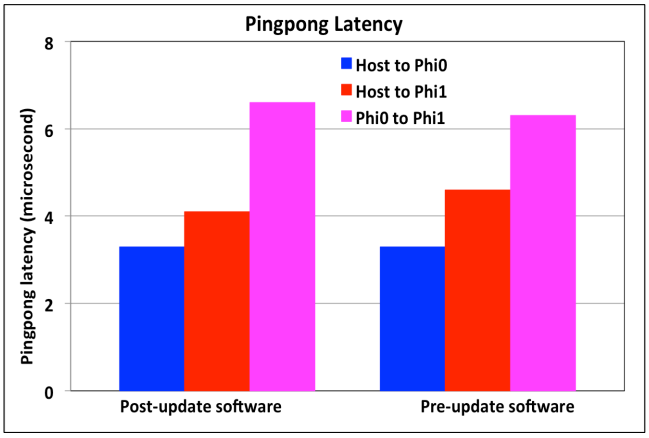

Figure 7. MPI latency between host and Phi.

6.3.2 MPI Bandwidth

MPI bandwidth for a 4-MB message size with the pre-update software was 1.6 GB/s, 455 MB/s, and 444 MB/s for host to Phi0, host to Phi1, and Phi0 to Phi1 respectively. These bandwidth values increased significantly with the post-update software to 6 GB/s, 6 GB/s, and 899 MB/s. The post-update bandwidth curves show three distinct regions corresponding to the three states discussed in Section 5 with a change in slope for messages between 8 KB and 256 KB. It should be noted that beyond a message size of 256 KB, the post-update software uses SCIF, which provides significantly higher bandwidth compared to the pre-update software, which uses the CCL direct DAPL provider.

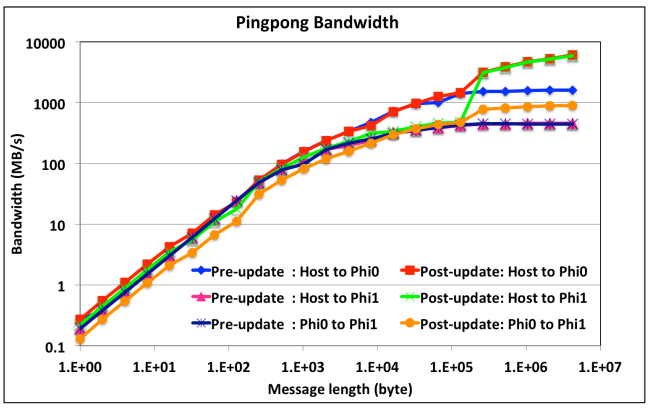

Figure 8. MPI bandwidth between host and Phi.

With pre-update software there is a performance asymmetry, i.e., the bandwidth of host to Phi0 (1.6 GB/s) is significantly higher than that for host to Phi1 (455 MB/s). The post-update software not only removed this performance asymmetry but also significantly increased the both bandwidth values to 6 GB/s. The post-update software also doubled the Phi0 to Phi1 bandwidth from 444 MB/s to 899 MB/s.

The improvement due to the post-update software is more visible in Figure 9, which plots the performance gain (ratio of post-update bandwidth to pre-update bandwidth) for host to Phi0, host to Phi1, and Phi0 to Phi1. For small to medium message sizes, the performance advantage of the post-update software is higher by a factor of 1 to 1.5 times and 1 to 1.3 for host to Phi0 and host to Phi1 respectively. For messages 256 KB or larger, where SCIF is used in the post-update software, the bandwidth is higher by a factor of 2 to 3.8 and 7 to 13 for host to Phi0 and host to Phi1 respectively. For Phi0 to Phi1 the bandwidth with post-update software decreased up to a message size of 8KB. However, for a message size of 256 KB or more, using SCIF improved the bandwidth by a factor of 1.8 to 2.

In summary, we can see that SCIF provides a significant increase in bandwidth for messages 256 KB or longer.

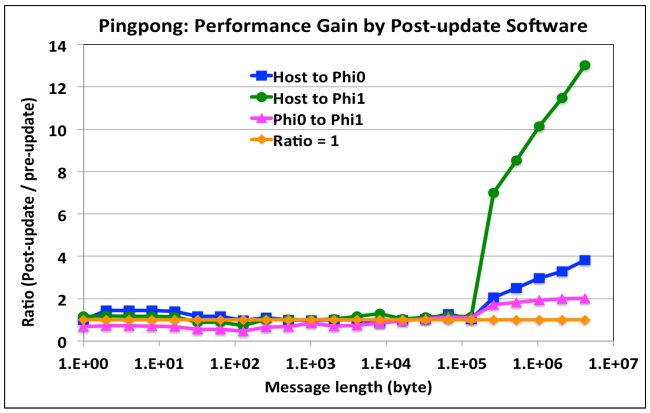

Figure 9. Performance gain in MPI bandwidth using post-update software.

6.4 MPI Functions

In this section we present the performance of selected MPI functions commonly used in NASA applications on the host and Phi coprocessors. In both cases we have used the same Intel compiler and Intel MPI library. Comparing the performance of MPI functions on both the intra-host and intra-Phi will give us insight into appropriateness of using the same library on two entirely different architectures.

6.4.1 MPI_Send/Recv benchmark

Figure 10 plots the point-to-point communication bandwidth achieved using an *MPI_Send/Recv* benchmark (each thread sends a message to its right neighbor and receives one from its left neighbor) for message sizes ranging from one byte to 4 MB using 1, 2, 3 and 4 threads per core on the Phi along with the corresponding results on the host for 16 threads. Performance on the host (16 threads) is higher than even one thread per core (59 threads) of the Phi by a factor of 1.3 to 3.5. For 4 threads per core (236 threads) performance of host is higher by a factor 24 to 54. For communication dominant code, it is beneficial to use only one thread per core on the Phi.

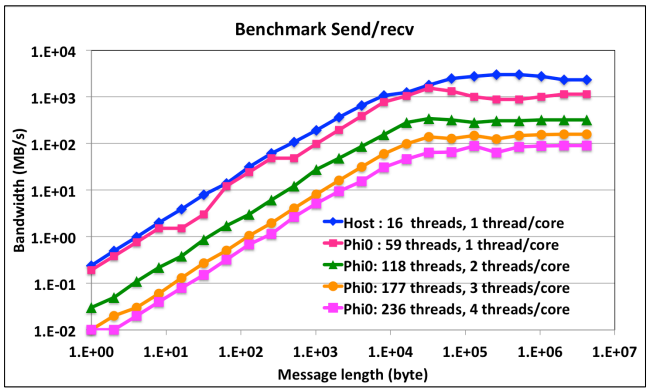

Figure 10. Performance of *MPI_Send/Recv* on host and Phi.

6.4.2 MPI_Bcast

Figure 11 shows the performance of *MPI_Bcast* on both the host and Phi0. This MPI function is used in the MPI version of Cart3D

where a message size of 56 MB is broadcast by the master process to all other processes. The performance of *MPI_Bcast* on the host is higher than on Phi0 with 1 thread per core (59 threads) by factor of 1.1 to 3.8. Per core performance on the host is higher by a factor of 20 to 35 than on Phi0 with 4 threads per core (236 threads). It is clear that using more than one thread per core decreases the performance drastically.

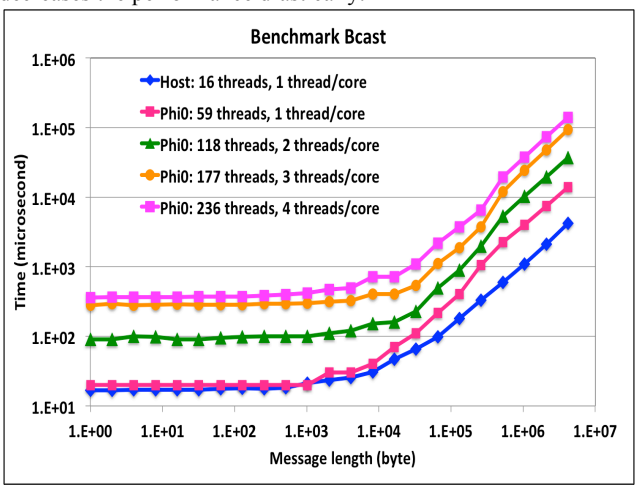

Figure 11. Performance of *MPI_Broadcast* on host and Phi.

6.4.3 MPI_Allreduce

In Figure 12 we present the performance of *MPI_Allreduce* for both host and Phi0. This is a key MPI function in NASA codes such as USM3D, MITgcm, and FUN3D, etc. Performance on the host is higher than on Phi0 (1 thread per core) by a factor of 2.2 to 13.4. The host performance is higher by a factor of 28 to 104 than on Phi0 with 4 threads per core.

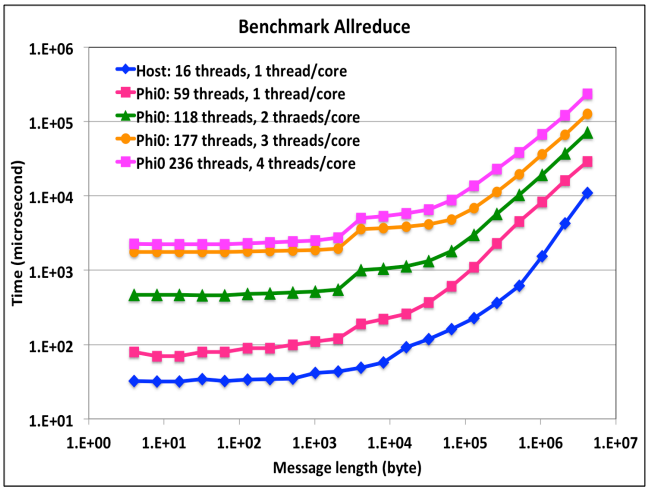

Figure 12. Performance of *MPI_Allreduce* on host and Phi.

6.4.4 MPI_Allgather

Figure 13 gives the results of *MPI_Allgather* on the host and Phi0. Performance of the host is always much higher than that on the Phi0. On Phi0, time increases smoothly as message size increases from 1 byte to 1 KB and then at 2KB and 4KB time increases abruptly and then again becomes smooth from 8KB onwards. This sudden jump in time at 2KB and 4KB message size is due to a change in algorithm used in *MPI_Allgather*. Performance on the host is higher than that on the Phi by a factor of 2.6 to 17.1 for one thread per core (59 threads) and by a factor 68 to 1146 for 4 threads per core (236 threads).

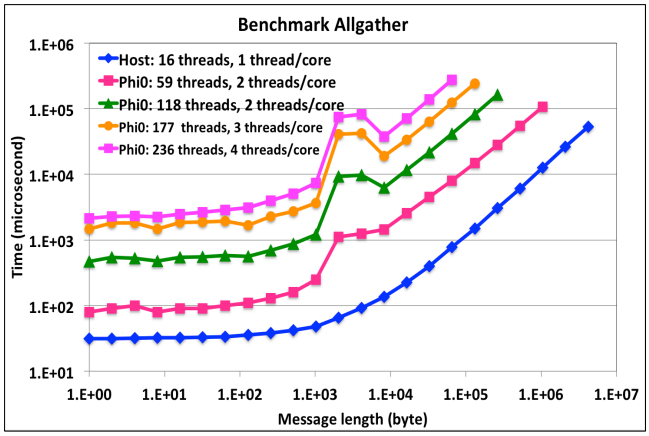

Figure 13. Performance of *MPI_AllGather* on host and Phi.

6.4.5 MPI_AlltoAll

In Figure 14, we present the results of *MPI_AlltoAll* for both the host and Phi0. This benchmark did not run successfully for all the message sizes from 1 byte to 4 MB. For 4 threads per core (236 threads) it could be run only up to a maximum message size of 4 KB. The failures were due to a lack of memory (as we also see in Figure 20 where we could not run the MPI version of NPB FT for the same reason). For one thread per core the performance of the host is higher than on Phi0 by a factor of 8 to 20, which is much higher than other forms of communications. For 4 threads per core on Phi0 (236 threads), host performance is higher by a factor of 1003 to 2603.

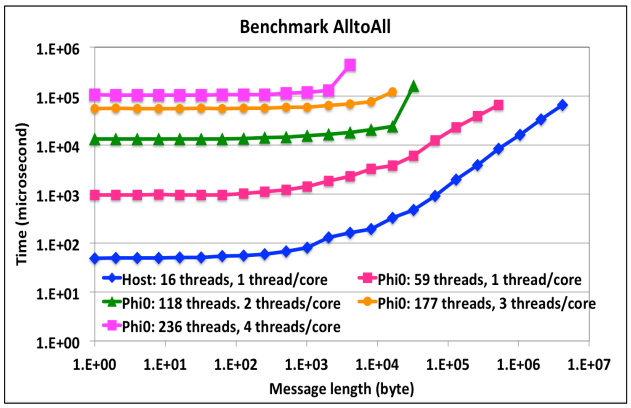

Figure 14. Performance of *MPI_AlltoAll* on host and Phi.

6.5 OpenMP Microbenchmarks

In this subsection we present results for synchronization and loop scheduling OpenMP benchmarks.

6.5.1 Synchronization

The synchronization OpenMP benchmark measures the overhead incurred by explicit barrier or implicit barrier at parallel and work sharing constructs, and by mutual exclusion constructs. Work-sharing constructs used in this test include DO/FOR, PARALLEL DO/FOR, and SINGLE. The mutual exclusion constructs include CRITICAL, LOCK/UNLOCK, ORDERED, and ATOMIC.

Figure 15 shows the synchronization overheads for several OpenMP directives on the host (16 threads on 16 cores) and Phi0 (59 cores or 236 threads, 4 threads per core). Overhead in terms of the sequential time T_s, and the parallel time T_p on p threads is $T_p - T_s/p$. We notice that almost all the constructs have almost an order of magnitude higher overhead on the Phi than on the host. The

most expensive operation is Reduction, followed by PARALLEL FOR and PARALLEL, whereas ATOMIC is the least expensive.

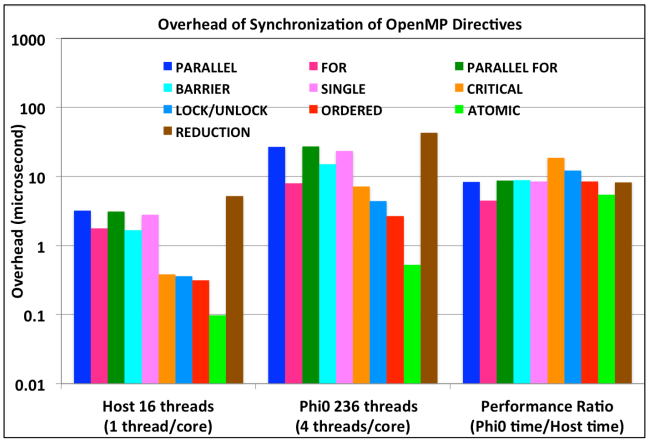

Figure 15. OpenMP synchronization overhead on host and Phi.

6.5.2 Loop Scheduling

Loop scheduling affects how the loop iterations are mapped onto threads. The scheduling benchmark measures the overheads for three scheduling policies: STATIC, DYNAMIC, and GUIDED.

Figure 16 presents the scheduling overheads on the host and the Phi. We find overhead on Phi is an order of magnitude higher than that on the host for all the three scheduling policies. As expected the STATIC overhead is the lowest, the DYNAMIC overhead is highest, and the GUIDED overhead is in between the two.

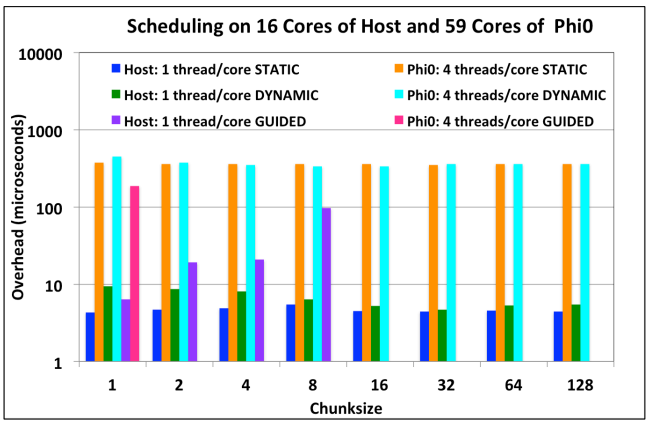

Figure 16. OpenMP scheduling overheads on host and Phi.

6.6 I/O benchmarks

Figure 17 presents the sequential read and write bandwidth for the host, Phi0, and Phi1. I/O benchmarks were run on a Network File System (NFS) mounted on the host. This NFS is exported to Phi0 and Phi1. Write bandwidth on the host and Phi0 is about 210 MB/s and 80 MB/s respectively. Read bandwidth is 295 MB/s and 75 MB/s for the host and Phi0. Write bandwidth on host is 2.6 times higher and read bandwidth 3.9 times higher than on Phi0. The poor performance of I/O in native Phi mode is due to the fact that read/write on Phi is done via the TCP/IP stack in MPSS over PCIe fabric resulting in a virtual network. Intel is working on a new TCP/IP stack that will address the issue in the future release of MPSS. Intel states that if an application has significant I/O, use of native Phi mode is not recommended [32]. As a workaround, this performance problem can be overcome by creating a new MPI process on the host and then sending the data from host to Phi or Phi to host by using MPI_Send/MPI_Recv via SCIF over the PCIe, which gets a bandwidth of 6 GB for message sizes of 4MB or more, and then perform read/write to the disk from the host [33].

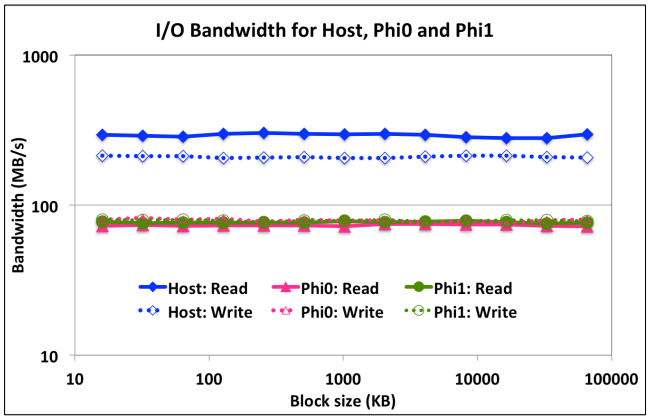

Figure 17. Read and write bandwidth on host, Phi0, and Phi1.

6.7 Offload Bandwidth over PCIe

Performance in the offload mode to a large extent depends on the data transfer bandwidth from host to Phi and vice versa over the PCIe bus. Therefore, it is useful to measure this PCIe bandwidth. A data packet sent via PCIe in offload mode has framing (start and end), a sequence number, a header, data, a digest, and a link cyclical redundancy check. To transmit 64 or 128 bytes of data, it has to be packed in 20 bytes of wrapping, yielding a maximum efficiency of 76% and 86% respectively, or 6.1 GB/s and 6.9 GB/s. We wrote a benchmark to measure this offload PCIe bandwidth between host and Phi for varying data sizes. Figure 18 shows the bandwidth of host to Phi. For large data transfers it is about 6.4 GB/s. Note that bandwidth from host to Phi0 is about 3% higher than for host to Phi1 for large data sizes. In addition, there is fall in bandwidth at a data size of 64 KB, which is currently not understood and needs further investigation.

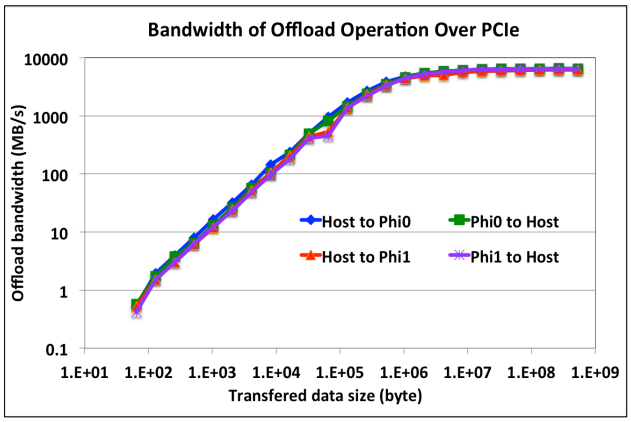

Figure 18. Offload bandwidth between host and Phi.

6.8 NAS Parallel Benchmarks (NPB)

In this section we present results for both OpenMP and MPI versions of the NPBs.

6.8.1 NPB OpenMP Version

Figure 19 shows the performance of the OpenMP version of the NPBs on the host and Phi0. Performance on the host is always better than the best possible performance on Phi0. Performance on Phi0 depends strongly on the number of threads on each core. In

native mode, performance on Phi0 is minimal for 1 thread per core and maximal for the 3 threads per core for most of the benchmarks. Among the six benchmarks, BT has the highest performance and CG the lowest on the Phi. The reason for this is that BT is vectorized, compute intensive, and highly parallel so it can use the 512-bit wide vector-unit and the hardware multithreading. CG finds the smallest eigenvalue of a symmetric positive definite matrix and uses indirect addressing. As such, it cannot reuse the cache efficiently. Our tests indicated that the compiler vectorized the most time consuming loop (sparse BLAS) using the gather-scatter vector instructions. However, performance of this version was only 10% better than the version without vectorization. This shows that the gather-scatter instruction is not efficient on Phi. Except for MG, most of the benchmarks have worse performance on the Phi than on the host.

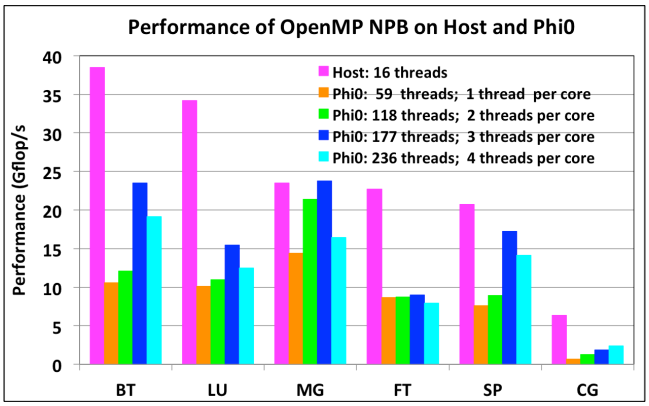

Figure 19. Performance of NPB OpenMP on host and Phi.

6.8.2 NPB MPI Version

Figure 20 shows the performance of the MPI version of the NPBs. The minimum and maximum numbers of threads available on a Phi are 59 and 236 respectively so results for CG, MG, FT, and LU are shown only for 64 and 128 MPI processes as these benchmarks run only using power of two processes. For BT and SP, the results are shown only for 64, 121, 169 and 225, as these benchmarks require square process grids. The FT benchmark could not be run on Phi because the Phi memory of 8GB is not enough, as it needs minimum of 10 GB to run with 64 or more processes. This brings some challenges to run MPI codes on the Phi, especially for those that have constraints on the number of MPI ranks. The other finding is that unlike in the OpenMP version, 3 threads per core do not always give the best performance, e.g., BT performance is best for 4 threads per core. It shows that number of threads per core needs some tuning to determine its optimal value for a given application.

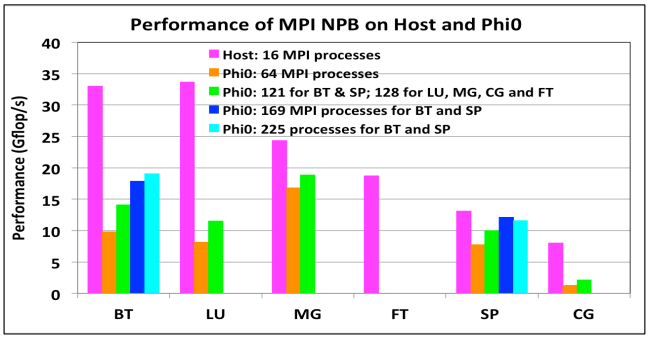

Figure 20. Performance of NPB MPI on host and Phi.

6.9 Science and Engineering Applications

In this subsection we focus on the comparative performance of two full production quality applications, OVERFLOW and Cart3D, on the host and the Phi coprocessor. We used a hybrid (MPI + OpenMP) version of OVERFLOW, and for Cart3D we used the pure OpenMP version. OVERFLOW was run in three modes: native host, native Phi and symmetric modes. Cart3D results are presented only for native host and native Phi modes, as a pure OpenMP code cannot run in symmetric mode. We also present results for MG of the NPB suite in offload mode.

6.9.1 Native Mode

In this subsection we present the results of Cart3D and OVERFLOW in native host and native Phi modes.

6.9.1.1 Cart3D

Figure 21 shows the performance of the OpenMP version of Cart3D in native host and native Phi modes. Native host results are shown for 16 OpenMP threads (one thread per core) whereas native Phi results are for 59, 118, 177, and 236 threads corresponding to 1, 2, 3 and 4 threads per core. Host performance is two times better than the best result on Phi. Performance on Phi is the best for 4 threads per core. As noted earlier, the number of threads per core is a tunable parameter and 4 is the optimal for Cart3D, unlike the NPBs where 3 is generally the best value.

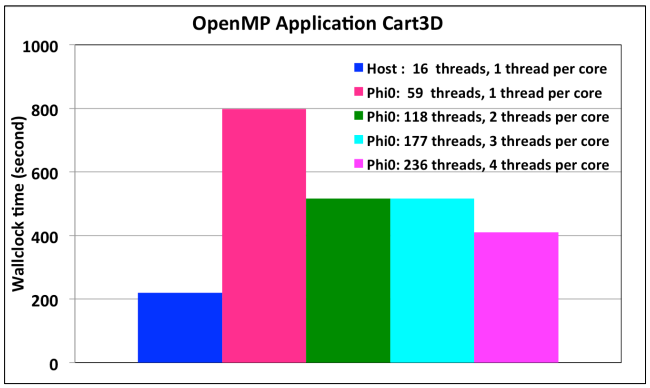

Figure 21. Performance of Cart3D on host and Phi.

6.9.1.2 Overflow

Figure 22 shows the wallclock time per step of the hybrid (MPI + OpenMP) OVERFLOW in native host and native Phi modes for the DLRF6-Medium data set. The notation (I x J) is used here, where I is the number of MPI processes and J is the number of OpenMP threads per MPI process. For example, 8 x 2 means 8 MPI processes and 2 OpenMP threads per MPI process. We used one host (two Sandy Bridge processors) and one Phi (Phi0).

The best performance on the host is for 16 x 1 whereas the worst performance is for 1 x 16. The best performance on the Phi is for 8 x 28 (224 threads—close to 4 threads per core) and the worst performance is for 4 x 14 (56 threads—close to 1 thread per core). The best performance on the Phi is worse than the best performance on the host by a factor of 1.8. On the host, performance decreases as the number of OpenMP threads increases for a fixed number of total threads. On the other hand, on the Phi, performance increases as the number of OpenMP threads increases. The main reason for the lower performance of OVERFLOW on the Phi is that the performance of OVERFLOW depends on the bandwidth of the memory subsystem, which is much lower on the Phi than on the host.

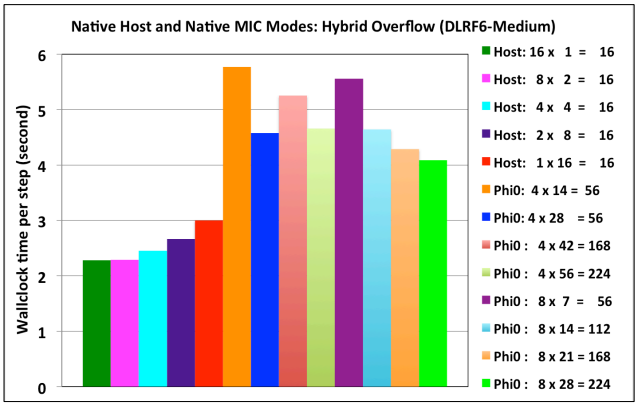
Native Host and Native MIC Modes: Hybrid Overflow (DLRF6-Medium)
Wallclock time per step (second)
Host: 16 x 1 = 16
Host: 8 x 2 = 16
Host: 4 x 4 = 16
Host: 2 x 8 = 16
Host: 1 x 16 = 16
Phi0: 4 x 14 = 56
Phi0: 4 x 28 = 56
Phi0 : 4 x 42 = 168
Phi0 : 4 x 56 = 224
Phi0 : 8 x 7 = 56
Phi0 : 8 x 14 = 112
Phi0 : 8 x 21 = 168
Phi0 : 8 x 28 = 224

Figure 22. Performance of Overflow on host and Phi .

6.9.1.3 Symmetric Mode

Figure 23 shows the wallclock time per step of OVERFLOW on the DLRF6-Large case in a symmetric mode on host+Phi0+Phi1 for pre-update and post-update software. Also shown in the figure is the percentage performance gain by the post-update software. In the symmetric mode there is MPI communication over PCIe amongst the host, Phi0, and Phi1, so the upgrade due to the post-update software does impact the performance of the application. The performance gain using post-update software is from 2% to 28%.

The best performance is obtained when one OpenMP thread is used for each MPI rank on the host and 28 OpenMP threads are used for each MPI rank on Phi0 and Phi1. The 8x28 case has 224 threads, which almost fills the Phi. Performance in this symmetric mode, which uses host, Phi0 and Phi1 with 8 MPI ranks on each Phi and 28 OpenMP threads for each MPI rank, is better than that on host only (not shown in the chart) by a factor of 1.9.

When compared to using two hosts (host1+host2), the best host+Phi0+Phi1 result is still worse. Detailed examination of the results revealed that the host+Phi0+Phi1 combination was about 15% faster than the two hosts on the numerically intensive parts of the code, but communication time and overhead due to load imbalance (which might include some communication time) were large enough on the host+Phi0+Phi1 to outweigh the speedup on the numerically intensive parts.

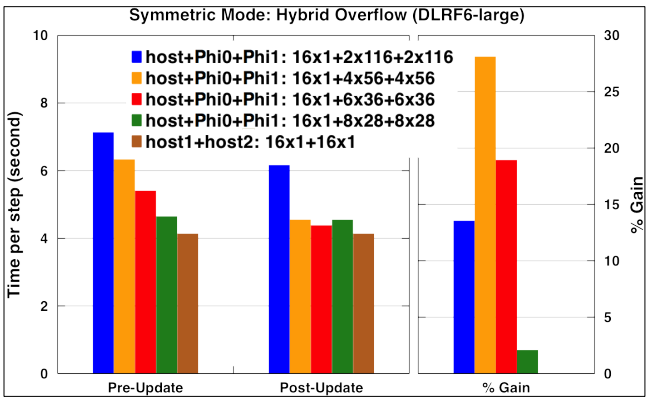

Figure 23. Performance of Overflow in symmetric mode.

6.9.1.4 Offload Mode

In this section we present the results for three different versions of the MG benchmark in offload mode and compare with native host and native Phi modes. Also presented is the cost of offload from host to Phi. All offload tests are done on Phi0. We used the Intel tool OFFLOAD_REPORT for producing the offload profile. The offload cost has three components:

- Setup time + data gather/scatter time on host
- PCIe transfer time
- Setup time + data gather/scatter time on Phi

6.9.1.5 OpenMP Loop Collapse

We ported the NPB OpenMP version of MG for offload testing on Maia. The MG benchmark approximates the solution to a three-dimensional discrete Poisson equation using the V-cycle multigrid method. The basic idea behind MG method is to reduce long wavelength error components by updating blocks of grid points. We present results from two versions, the original benchmark from the NPB version 3.3 suite and an optimized version in which the OpenMP nested loops were collapsed. This loop-collapsing optimization increased the performance by 25% to 28% on Phi0 as shown in the Figure 24. However, this optimization degraded the performance on the host for 16 threads by 1%, showing that a transformation good for Phi is not necessarily best for the host.

Note that performance with 59, 118, 177, and 236 threads is much better than with 60, 120, 180, and 240 threads, respectively. The reason for this is that 59, 118, 177, and 236 use 59 cores with 1, 2, 3, and 4 threads per core respectively. Using the 60th core, which is usually used for OS services, incurs significant overhead, and should be avoided.

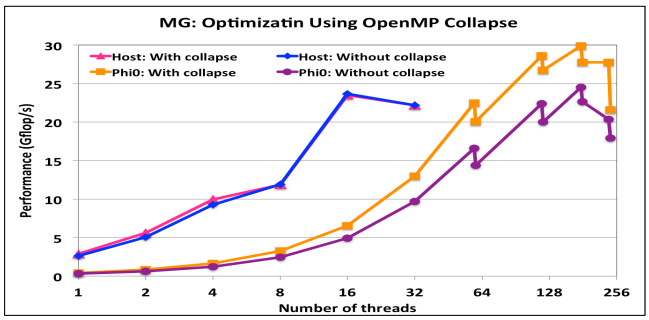

Figure 24. Performance gain of OpenMP loop collapse on Phi.

6.9.1.6 Native and Offload Modes

Figure 25 shows the performance of MG in native host, native Phi, and offload modes. For offload we used three different versions:

- offload one OpenMP loop,
- offload one subroutine in the main program and
- offload the whole computation.

The performance of MG in native host mode (23.5 Gflop/s for 16 threads) is lower by 27% than the best performance on the Phi (29.9 Gflop/s for 177 threads – 3 threads per core) in native Phi mode. Note that on the host, HT performance (32 threads – 22.2 Gflop/s) is 6% lower than single thread performance (16 threads – 23.5 Gflop/s). We see that the performance of all the offload versions is much lower than both native host and native Phi modes. The main reason for this is the high cost of data transfer.

The amount of data transferred between host and Phi, and the number of offload invocations are different for the three offload versions. We offloaded the most time consuming "do loop" in the subroutine "resid". Here the amount of data transferred is the least in each offload occurrence. But the total amount of data transferred and the number of offload invocations are the most among all three versions. So the performance of this version is the worst. The performance is improved when offloading the whole

subroutine "resid" with fewer offload occurrences and data. The most efficient technique is to offload the whole computation to the Phi. In this version the data transferred is the least because input data is generated on host and transferred to Phi only once. So the main criteria to evaluate whether an application is suitable for offload mode is the cost of data transfer and offload overhead. It is clear from our study that MG is not a good candidate for offload mode.

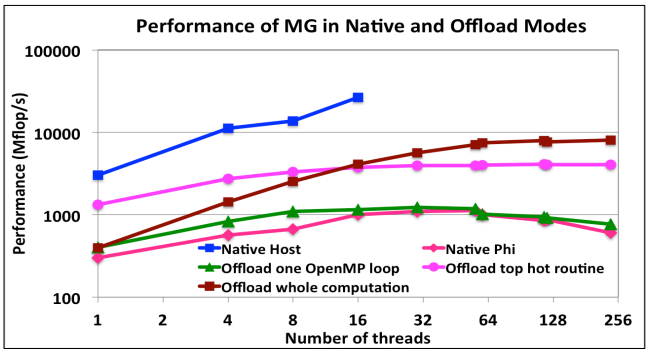

Figure 25. MG in 3 modes: native host, native Phi, offload.

6.9.1.7 Overhead in Offload Mode

Figure 26 shows the overhead for the three offload versions of MG for 3 threads per core. As can be seen from this figure, the performance of offloading one main OpenMP loop is the worst and the best performance is that of offloading the whole computation as it has the least amount of data transferred between host and Phi.

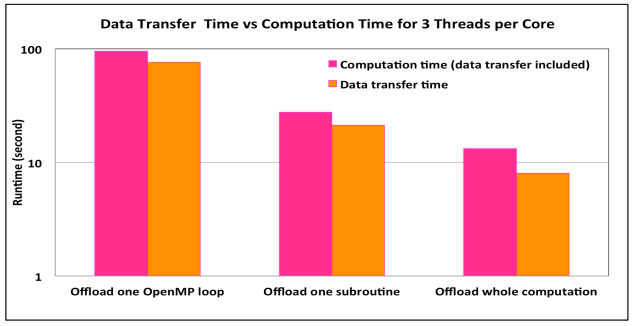

Figure 26. Overhead in three offload versions for MG.

Figure 27 shows the number of offload invocations and the amount of data transferred in the three versions of the offload code. This cost is maximal for offloading one OpenMP loop and minimal for offloading the whole computation.

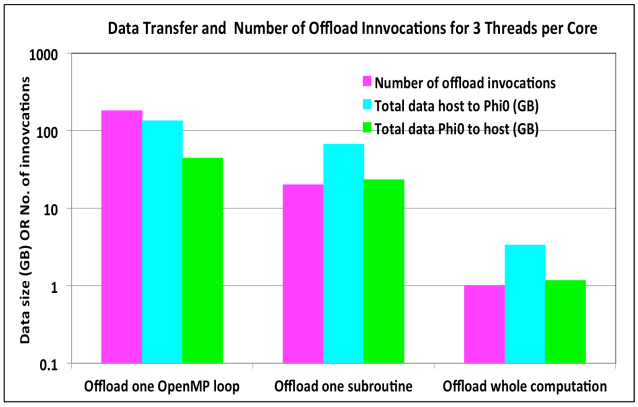

Figure 27. Cost of three offload versions of MG.

7. CONCLUSIONS

In this paper we studied the single node performance of an SGI Rackable computer that has Intel Xeon Phi coprocessors. We ran a suite of codes ranging from microbenchmarks to full CFD applications of interest to NASA—Cart3D, which is pure OpenMP and OVERFLOW, which is an MPI+OpenMP hybrid code. We tested four programming modes: processor native, coprocessor native, symmetric, and offload. After we finished our initial experiments, an updated version of software became available and we repeated the runs to determine its impact.

When comparing native performance of the two applications, we found that a single Phi card had about half the performance of the two host Xeon processors. When run in symmetric mode with two Phi coprocessors, OVERFLOW achieved a 1.9x boost compared to its best performance in native host mode.

The advantage of native mode on Phi is that the code requires no changes. On the negative side, there are constraints on memory footprint, and I/O performance is poor due to NFS mounting being done via TCP/IP on PCIe. In contrast, symmetric execution requires careful balancing of the workload across the host processors and the Phi coprocessors.

Our less-than-optimal application performance results can be explained as follows. Peak performance on Phi requires a highly threaded code that is also highly vectorized with unit stride. This will keep the 512-bit wide vector units busy. Cart3D is not heavily vectorized. Furthermore as our microbenchmarks results show, the Phi has higher memory latency than Sandy Bridge, which results in additional stall cycles, and it has lower memory bandwidth, which starves the floating-point units. This limits the performance of memory bandwidth-intensive applications such as OVERFLOW.

As our experiments show, there is a significant overhead to using the offload mode and thus one should carefully choose the granularity of the offloads to offset the overhead of the data transfer with the efficiency gained by execution on the coprocessor.

We found that the overhead of system software such as MPI and OpenMP is very high on Phi and needs to be optimized. In addition, better performance can often be achieved by leaving one core to operating system software when running a user application on Phi. In addition, the implementation of gather and scatter on the Phi is not efficient as is shown by the non-unit stride vectorization of CG and OVERFLOW.

The post-update software significantly enhanced the MPI bandwidth over PCIe especially for large message sizes and performance of OVERFLOW. Software on the Phi is maturing gradually and the next generation of hardware is expected to be promising for achieving high performance on highly vectorized and highly parallel applications.

The less than hoped for application performance described in the paper is a combination of hardware (low memory bandwidth, light core, small memory, network latency and bandwidth, etc.), software (MPI library, operating system, and compiler), and applications. On the positive side, we have seen that the software is evolving and improving; we also note that some applications can be reformulated. We hope that the hardware issues will be resolved in the next version of the Phi to provide a better performing system.

8. ACKNOWLEDGEMENTS

The authors are grateful to Matt Reilly of Institute for Defense Analyses for several valuable suggestions. Valuable support and

help of Johnny Chang and John Hardman is gratefully acknowledged. Work by Jahed Djomehri, William Arasin, and Robert Hood, employees of CSC, was supported by NASA Contract No. NNA07CA29C.

9. REFERENCES

[1] Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-Stover. GPU cluster for high performance computing. In SC'04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, page 47,Washington, DC, USA, 2004. IEEE Computer Society.

[2] Jens Krüger and Rüdiger Westermann. Linear algebra operators for GPU implementation of numerical algorithms. ACM Transactionson Graphics (TOG), 2 2(3): 908–916, 2003.

[3] John Michael Levesque, Grout Ray, Ramanan Sankaran, Hybridizing S3D into an Exascale Application using OpenACC, In SC'12: Proceedings of the 2012 ACM/IEEE Conference on Supercomputing, Salt Lake City, 2012, IEEE Computer Society.

[4] *Intel Xeon Phi Coprocessor Developer's Quick Start Guide,* http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide.

[5] *Intel Xeon Phi Coprocessor (codename Knights Corner),* http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner.

[6] Introducing Titan: Advancing the Era of Acceleeatred Computing, http://www.olcf.ornl.gov/titan/.

[7] About the *Blue Waters* project, www.**ncsa**.illinois.edu/**BlueWaters**/

[8] Texas Advanced Computing Center - *Stampede*, http://www.tacc.utexas.edu/**stampede.**

[9] New supercomputer coming to EMSL this summer, supplied by Atipa Technologies, http://www.pnnl.gov/news/release.aspx?id=965, March 2013.

[10] 10 Petaflops Supercomputer "Stampede" Powered by Intel® Xeon Phi™ Coprocessors Officially Dedicated Today, http://newsroom.intel.com/community/intel_newsroom/blog/2013/03/27/10-petaflops-supercomputer-stampede-powered-by-intelr-xeon-phitm-coprocessors-officially-dedicated-today, March 27, 2013

[11] TACC-Intel Highly Parallel Computing Symposium – Preaparing for Many Core, Knight Ferry, http://www.tacc.utexas.edu/news/feature-stories/2012/preparing-for-many-core, April 10-11, 2012.

[12] Early Application Experiences with the Intel® MIC Architecture, Electronic Structure Calculation Methods on *Accelerators*. *Workshop* – Oak Ridge, TN – February 6 - 8, *2012.*

[13] T.G. Mattson, R. Vander Wijngaart, and M. Frumkin, "Programming the Intel 80-core network-on-a-chip tera scale processor," in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing,. Piscataway, NJ, USA: 2008, pp. 38:1–38:11 http://dl.acm.org/citation.cfm?id=1413370.1413409

[14] SGI with *Intel Xeon Phi* Coprocessors, www.sgi.com/products/servers/accelerators/**phi**.html.

[15] Intel® Xeon® Processor E5 Family, http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-5000-sequence.html.

[16] S. Saini, A. Naraikin, R. Biswas, D. Barkai and T. Sandstrom, "Early performance evaluation of a "Nehalem" cluster using scientific and engineering applications", Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2009, November 14-20, 2009, Portland, Oregon, USA.

[17] S. Saini, H. Jin, R. Hood, D. Barker, P. Mehrotra and R. Biswas, "The impact of hyper-threading on processor resource utilization in production applications", **Best Paper**, 18th International Conference on High Performance Computing, HiPC 2011, Bengaluru, India, December 18-21, 2011.

[18] STREAM Version 5.10: Sustanable Memory Bandwidth in High Performance Computers: http://www.cs.virginia.edu/stream/

[19] D. Molka, D. Hackenberg, R. Schöne and M. S. Müller, *Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System*, In Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques (PACT'09), pages 261-270, IEEE, 2009, http://ieeexplore.ieee.org/search/srchabstract.jsp?tp=&arnumber=5260544

[20] R. Schöne, D. Hackenberg, D. Molka, "Memory performance at reduced CPU clock speeds: an analysis of current x86_64 processors", In Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems (HotPower'12), October 7, 2012, Hollywood, USA http://dl.acm.org/citation.cfm?id=2387869.2387878

[21] S. Saini, R. Ciotti, B. TN Gunney, T. E. Spelce, A. Koniges, D. Dossa, P. Adamidis, R. Rabenseifner, S. R. Tiyyagura, and M. Mueller. "Performance evaluation of supercomputers using hpcc and imb benchmarks." *Journal of Computer and System Sciences* 74, no. 6 (2008): 965-982.

[22] James LaGrone, Ayodunni Aribuki, and Barbara Chapman, A set of microbenchmarks for measuring OpenMP task overheads. International Conference on Parallel and Distributed Processing Techniques and Applications, vol II, pp. 594-600, (July 2011).

[23] Bronis R. de Supinski, *LLNL* OpenMP Performance Suite Description, https://computation.**llnl**.gov/casc/**sphinx**/, *2001*

[24] J. Mark Bull, Fiona Reid, Nicola McDonnell: A Microbenchmark Suite for OpenMP Tasks. IWOMP 2012: 271-274.

[25] S. Saini, D. Talcott, R. Thakur, P. Adamidis, R. Rabenseifner, and R. Ciotti. "Parallel I/O performance characterization of Columbia and NEC SX-8 superclusters." In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1-10. IEEE, 2007.

[26] NAS Parallel Benchmarks (NPB), http://www.nas.nasa.gov/publications/npb.html

[27] OVERFLOW, http://aaac.larc.nasa.gov/~buning/

[28] S. Saini, D. Talcott, D. Jespersen, J. Djomehri, H. Jin, and R. Biswas. "Scientific application-based performance comparison of SGI Altix 4700, IBM POWER5+, and SGI ICE 8200 supercomputers." In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pp. 1-12. IEEE, 2008.

[29] S. Saini, P. Mehrotra, K. Taylor, S. Shende and R. Biswas, Performance Analysis of Scientific and Engineering Applications Using MPInside and TAU, pp. 265-272, in: Proc. 12th IEEE Intl. Conf. on High Performance Computing and Communications, Melbourne, Australia, 2010.

[30] D. J. Mavriplis, M. J. Aftosmis, and M. Berger. High Resolution Aerospace Applications using the NASA Columbia Supercomputer, in: Proc. ACM/IEEE SC05, Seattle, WA, 2005.

[31] InfiniBand Trade Association: http://www.infinibandta.org.

[32] Building a Native Application for Intel® Xeon Phi™ Coprocessors, http://software.intel.com/en-us/articles/building-a-native-application-for-intel-xeon-phi-coprocessors.

[33] I/O performance and best way to run Native executable, http://software.intel.com/en-us/forums/topic/382695.